Robustness via run-time adaptation of contingent plans

John L. Bresina and **Richard Washington**[*]
NASA Ames Research Center, M/S 269-2
Moffett Field, CA 94035
{jbresina, rwashington}@arc.nasa.gov

Abstract

In this paper, we discuss our approach to making the behavior of planetary rovers more robust for the purpose of increased productivity. Due to the inherent uncertainty in rover exploration, the traditional approach to rover control is conservative, limiting the autonomous operation of the rover and sacrificing performance for safety. Our objective is to increase the science productivity possible within a single uplink by allowing the rover's behavior to be specified with flexible, contingent plans and by employing dynamic plan adaptation during execution. We have deployed a system exhibiting flexible, contingent execution; this paper concentrates on our ongoing efforts on plan adaptation.

Plans can be revised in two ways: plan steps may be deleted, with execution continuing with the plan suffix; and the current plan may be merged with an "alternate plan" from an on-board library. The plan-revision action is chosen to maximize the expected utility of the plan. Plan merging and action deletion constitute a more conservative general-purpose planning system; in return, our approach is more efficient and more easily verified, two important criteria for deployed rovers.

Introduction

In this paper, we discuss our approach to making the behavior of planetary rovers more robust for the purpose of increased productivity. There is a significant degree of inherent uncertainty in rover planetary exploration with respect to resource availability and usage, task duration, and degree of successful task execution. In addition, there is scientific uncertainty; that is, uncertainty in what instrument readings might reveal and, hence, which targets are going to be the most important. The traditional approach to handling all such sources of uncertainty is to uplink conservative command sequences and limit what can be performed autonomously by a rover. Rover behavior is rigidly governed by the uplinked sequence of commands, and when execution falls outside this narrow scope of allowed behavior, the rover must sit idle awaiting a new sequence until the next communication opportunity.

Our objective is to increase the science productivity possible within a single uplink by allowing the rover's behavior to be specified with flexible, contingent plans and by employing dynamic plan adaptation during execution. In our approach, plans have flexible temporal windows on each command and can include contingent branches conditioned on the runtime environment. A contingent plan can be thought of as a tree of possible behaviors that is planned in advance to account for possible failures that might occur or opportunities that may arise during execution. Contingent branches can also represent disjunctive choices, in which case the rover must choose the best branch at execution time. In order to intelligently make this decision, the plan includes utility estimates based on the value of the plan's tasks and the probability that the tasks will be successfully executed.

During execution, the overall utility of the uplinked plan may change due to unforeseen science opportunities or changes in resource availability and demand profiles. In this case, the rover's behavior will be suboptimal unless its plan is revised. Furthermore, the scope of the plan may be exceeded due to failures. In this case, without plan revision, the rover must wait until a new plan is uplinked. We address the issues of plan suboptimality and plan failures by performing limited plan revision on board. In particular, plans can be revised in two ways: plan steps may be deleted, with execution continuing with the plan suffix; and the current plan may be merged with an "alternate plan" from an on-board library. The plan-revision action is chosen to maximize the expected utility of the plan. If plan revision cannot increase the expected utility of the current plan, it remains unchanged.

Plan merging and action deletion constitute a more conservative plan-revision approach than one that employs an on-board general-purpose planning system,

[*] NASA contractor with RIACS.

which can replan from scratch. Plan merging and action deletion are less complex and, hence, more efficient, which is an important advantage given the limited computational resources of planetary rovers. Furthermore, our approach poses an easier software verification problem, which is advantageous in gaining the confidence and acceptance of flight operators. Our long-term strategy is to continue to increase rover on-board planning capabilities as flight processors become more powerful and as rover autonomy software becomes more acceptable.

In the next section, we describe some of our previous work that serves as the background to the current, ongoing extensions which are the focus of this paper. We then present an illustrative example of the robust behavior we hope to enable, followed by a discussion of our proposed approach to run-time plan adaptation. The last two sections describe some related work and discuss our effort's current status and open issues.

CRL and Architecture

We have developed a flexible, contingent plan language that can represent a large family of valid execution behaviors and an executive that incrementally selects and carries out the most appropriate behavior (from the uplinked plan) in response to the rover's internal status and its interaction with the environment. The architecture achieves robust operation through the executive's interactions with the state identification, resource management, and plan adaptation mechanisms (each of these is discussed below). For further details on this background material, see (Bresina *et al.* 1999; Washington, Golden, & Bresina 1999; Washington *et al.* 1999)

The plan language used by the ground planning tools and on-board execution is the Contingent Rover Language (CRL). CRL was designed to serve as the communication medium between the ground operations team and a planetary rover, providing a flexible, contingent language that remains simple enough for planning and verification and compatible with existing command languages.

A CRL command plan contains a nominal sequence with a set of contingent branches, as well as a library of alternate plans (which are discussed in more detail below). If there are no deviations from the *a priori* execution expectations, then the rover's behavior is governed by the nominal sequence. The contingent branches specify alternative courses of action in response to expectation deviations. Within any contingent branch there may be further contingent branches; hence, the plan is a tree of alternative courses of action.

To retain simplicity for planning and verification, CRL does not include any control constructs for looping. The design decision we made is that when a control loop is needed for execution robustness, it should be embedded within the implementation of a high-level CRL command.

The plan representation in CRL is a hierarchical, branching structure. The basic data type in CRL is a node; the subtypes provide the mechanisms for hierarchies and branching. CRL has three node subtypes: *block*, *task*, and *branch*; a command plan is defined to be a node, typically of subtype block. A *block* represents a sequence of nodes, over which there may be shared state conditions. Nested blocks provide the hierarchical structure of plans.

A *task* represents an action to execute. A task also specifies what action to perform if the task is interrupted due to execution failure. In addition, a task specifies a relative priority and expectations about resource and time usage. A *branch* represents a choice point in the command plan. Each of the execution paths is represented by an *option*. Nested branches lead to a tree of execution paths. An *option* is not a node subtype but a separate data type that has one subtype: *alternate plan*. Options and alternate plans specify the conditions under which they are eligible for execution and the node (typically of subtype block) to execute.

Each node has associated with it a set of conditions that must be satisfied for successful execution; the following are the condition types.

- *start-conditions*: The set of conditions that must be true for the node to begin execution. Conditions can include information about the internal state of the rover (*e.g.*, wheel current), external state (*e.g.*, location), and time windows.

- *wait-for-conditions*: A subset of start-conditions for which the rover will wait until they become true. Other conditions will fail without waiting. Some conditions are automatically waited for whether or not that is specified explicitly; *e.g.*, a constraint on when an action can start executing.

- *maintain-conditions*: A list of conditions that must be true throughout node execution

- *end-conditions*: A list of conditions that must be true at the end of node execution, to verify that an action had the intended effects. Constraints on action duration can be included here.

The rich expressiveness of temporal and other state constraints on the plan supports effective specification of science goals and safety policies, as well as providing

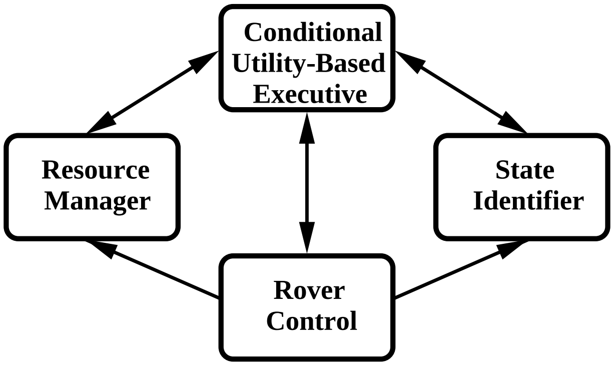

Figure 1: On-board Architecture.

increased flexibility during execution. For example, rather than time-stamps, each action can have a start time interval (and an end time interval).

A node also includes information regarding the expected utility of executing the rest of the plan, as well as information regarding how to react to execution failures: execution may continue to the next node or abort.

Our current on-board autonomy architecture, illustrated in Figure 1, consists of the following four components: a conditional, utility-based executive, the resource manager, the state identifier, and the rover control component.

The executive is responsible for interpreting the command sequence coming from ground control, checking run-time resource requirements and availability, monitoring plan execution, and potentially selecting alternative plan branches and modifying the plan if the situation changes.

The rover control is responsible for real-time execution of motor and sensing commands from the executive, as well as continual low-level state monitoring.

The resource manager receives estimated task resource profiles, monitors current and planned resource usage, and notices deviations from resource expectations. The rover executive adapts its actions in conjunction with the resource manager.

The state identifier receives low-level state information from the rover control component and uses its models of rover mechanics and behavior to identify the qualitative system state used by the executive. In addition, the state identifier is responsible for detecting anomalous states and notifying the executive about them. Our approach to state identification is to combine continuous probabilistic state estimation using Kalman filters (Grewal & Andrews 1993) with discrete qualitative state estimation using a Markov-model representation. This is an ongoing effort, and has not yet been integrated into the rover software.

In our current rover executive, plan execution proceeds by verifying conditions on each node and executing it when resource, time, and state constraints allow. At branch points in the plan, execution proceeds on the enabled branch with highest utility. This strategy allows the rover to operate in a large number of possible situations from a single uplinked plan. The initial estimate of the utility of executing actions is computed on the ground with respect to the expected resource and time availability. These estimates are only approximate — the actual time and resource availability is only known at execution time.

To better handle uncertainties, we have developed methods to update plan utilities on board at runtime to reflect the current best estimate of action utilities (Bresina & Washington 2000). Because the actions in our plan language can start within a flexible temporal interval, the expected utilities of the contingent options depend on the time that the branch point is reached during execution. Hence, a single utility measure is insufficient, and we need to compute a utility distribution that maps possible action start times to the expected plan-suffix utility, *i.e.*, the expected utility of executing the plan suffix starting with that action. The plan-suffix utility distribution is computed by forward propagation, to calculate time windows when actions could possibly be executed according to resource and temporal constraints, followed by backward propagation of the utility of individual actions, which, when combined with the probability of success and failure of actions, provides the expected utility. More details can be found in (Bresina & Washington 2000).

Expected plan-suffix utility depends on when actions can execute and with what probability. The time over which an action executes and the probabilities of success and failure are affected by all the constraints in the action's conditions (pre-, maintain, and end), as well as by the inherent uncertainty in action durations. As plan execution proceeds, the temporal windows for plan actions narrow, resource availability can change, and rover state can change in unpredictable ways. Such changes affect the execution time and success probabilities and, thus, the expected utilities. Note, however, that even though temporal changes can affect the probabilities of when future actions will start, the plan-suffix utility distributions of these actions do not have to be recomputed because they are conditioned on start time. Although the use of utility distributions does reduce utility recomputations, it does not eliminate them; *e.g.*, changes in resource availability can require dynamic utility updates.

Example

As an illustration of the type of problem where plan modification is appropriate, we present an example from the domain of rover exploration and show how the rover could react in this situation.

Imagine a scenario where scientists have identified an interesting rock near the rover. They construct a plan for the rover, where the primary goal is to acquire information about the chemical composition of the rock, followed by the transmission of the results to Earth. The rover's instruments include an arm-mounted spectrometer, another spectrometer mounted on the pan-tilt head on the central mast, and high-resolution science cameras also on the pan-tilt head. The expected quality of information from each instrument is decreasing in the order presented, as is the likelihood of failure to acquire data.

The goal of acquiring information is first decomposed into data acquisition *via* the arm-mounted spectrometer and spectral analysis. These subgoals are achieved by placing the arm on the rock, taking readings with the spectrometer, and analyzing those readings. However, there are a number of possible failures that could occur, including:

- the rover may not succeed in driving close enough to the rock,
- arm placement on the rough surface of the rock may fail to make contact close enough to the surface normal for spectrometry,
- the spectrometer itself may malfunction,
- the rover may run too low on energy.

Although a reasonable first attempt to recover from the failure may be to retry the action that failed, repeated failures may suggest that an alternative approach may be preferable. For example, if the rover is unable to deploy the arm or reach the rock, the mast-mounted spectrometer is a more appropriate backup instrument. If the energy available is too low for either of those instruments, the science cameras remain as the only reasonable instrument to use.

Suppose in executing the plan, the rover tries to place the arm on the rock but fails. After multiple attempts, the rover decides to use its mast-mounted spectrometer. After taking spectral data, the rover analyzes the data on board. A fortuitous result of the analysis is that a signature of carbonates is found in the spectral signal. Based on this finding, further spectral measurements and high-resolution images are taken to send to Earth for confirmation by scientists.

The above hypothetical execution trace involves a number of decisions taken reactively and opportunistically. The decision of how many times to try the arm placement could be a hard-coded number, but a better solution would be to make it depend on the expected success and expected quality of the resulting information. At some point, the diminishing hope of successful arm placement would cause the rover to abandon arm-based spectrometry in favor of a mast-based spectrometer reading. An opportunistic decision would arise from finding carbonate data; this would introduce a new goal, to confirm the reading, into the rover's plan.

Although this plan could be constructed as a complex conditional structure, a more compact and more general approach to solving this type of problem is to use utility-based plan modifications based on a library of plan fragments. Our approach is designed around this idea.

Run-time plan adaptation

Utility distributions allow the rover to choose the best course of action within its current plan, but this is often not sufficient due to the dynamic environment. The size and complexity of the "universal plan" necessary to handle all possible events renders it impractical to build, verify, and transmit to the rover. Instead, our approach is to adapt the uplinked plan during execution. The objective of this plan revision is to increase the utility of the plan and, hence, the productivity of the rover. For purposes of computational efficiency and verifiability, we propose to limit plan revision to two types of operations: skipping steps of an existing plan, and merging plans from an "alternate plan library" with the existing plan. The approach described here is work in progress, and we expect to learn from our experience in implementing and testing the approach on the real-world problem of controlling a rover performing scientific exploration.

The execution system will skip a node of the plan when the expected utility, at the current time instant, of executing the plan suffix starting with the current action is less than the expected utility of executing the plan suffix starting with the following action. The step skipped may be an executable action, or a higher-level block in the hierarchy. Since some nodes in the plan may depend on previous nodes, removing the current node may imply removal of some future nodes. For example, if the energy level of the rover after performing spectrometer readings is dangerously low, the analysis may be skipped in favor of simply sleeping until data communications time, sending the raw data back to Earth. In that case, the analysis routines would be skipped, and operations to queue the analysis results for downlink would also be deleted as a result.

Due to the removal of dependent actions, the expected utility of the plan suffix starting with the following action may change if the current step is skipped. The revised expected utility will need to be computed, either in advance or at run time, to evaluate the alternative courses of action. Computing the expected utility of a modified plan can be performed efficiently by propagating the change in utility backwards from removed steps to the current point in the plan.

The other mechanism for plan modification is to use alternate plans. An alternate plan specifies a global contingency for execution, as opposed to the local contingent and disjunctive branches at specific points within the plan. Alternate plans may be used to cover global failures, such as mechanical malfunction, instrument recalibration, and target loss, as well as global opportunities such as science discoveries or resource surpluses. In our example, obvious places for alternate plans would be instrument retries, backup methods for achieving goals, and opportunistic data collection. The impact of global branches is to increase the plan's effective branching and, thus, its coverage.

In contrast to local branches, the global branches are considered whenever their eligibility conditions are satisfied. During execution, when there are eligible alternate plans, the executive must choose between the following three options: simply continue with the existing plan, execute an alternate plan followed by continuation of the existing plan, or replace the rest of the existing plan with an alternate plan. The expected utility of all the options must first be estimated, then the executive will choose the most promising option.

An alternate plan may be applicable to a condition that arises anywhere within the active plan hierarchy. For example, a maintenance condition that applies to a plan block may be violated, and an alternate plan could be invoked to reestablish the condition rather than failing the entire block of execution and losing the execution context. Alternatively, an alternate plan could apply to a single executable action, for example, to continue a command that was suspended to handle an anomalous power state.

Since an alternate plan may contain actions that are either redundant or incompatible with the current plan, the execution system will modify the current plan, as described above for skipping steps, in order to find the highest-utility plan resulting from merging the alternate plan with the existing plan. Additionally, if the alternate plan serves to re-establish maintenance conditions that were lost, then the current action, or set of actions, may need to be re-executed after the alternate plan. For example, if the rover tries and fails to place the arm on a rock, then an alternate plan may reposition the rover with respect to the rock, from which point, the rover may retry the arm placement. We retain the goal structure of the plan to indicate the possible "restart" points within the plan. The plan may continue from the current point in execution, or it may back up to the beginning of any subgoal to which the failed action belongs. In our example, the arm placement retries and the backup data collection methods are all in service of the same goal, so execution would restart the achievement of the goal with each new action proposed. If too much time elapses during the retries, then the rover might need to skip ahead to its communication with Earth, at which point the goal has been lost and will not be reachieved, unless it is re-specified by ground scientists in a subsequent plan.

The alternate plans are individually verified on the ground with respect to the mission flight rules; however, additional on-board verification is needed during plan merging to ensure there are no harmful interactions between the existing plan and the alternate plan. For this purpose, constraints that are required for plan validity and flight safety will be encoded as annotations in alternate plans, as well as in uplinked plans. As alternate plans are first considered and then merged into the executing plan, the constraints may identify conflicts that will change the applicability of planned actions, potentially even eliminating them from the final plan. The final utility of the resulting plan will determine whether the modified plan is preferable to the unmodified plan or to another plan modification.

Related Work

A few areas of planning research have dealt with similar, although not identical, aspects of planning and plan adaptation. Differences in planning frameworks, representations, and application domains have led to the differences between these approaches and our own.

Plan merging in classical planning (Tsamardinos, Pollack, & Horty 2000; Yang 1997) is concerned with finding a global plan to achieve multiple goals by combining separate plans for each goal. In our framework, the notion of goal achievement (which is all-or-none) is replaced by utility (which is continuous). In our framework, plans that fail to achieve one or more goals would have reduced utility, whereas in the classical approach, such plans would have 0 utility and others would have either equivalent or cost-dependent utility. Resolving precondition-postcondition dependencies, as in the classical STRIPS representation, does not by itself lead to plans of maximal utility.

Yang (Yang 1997) addresses "optimization-based plan selection," using a heuristic approach to find the minimal-cost plan by merging separate plans. Once again, this is designed to find a plan that achieves multiple goals with minimal cost, as opposed to maximizing overall plan utility while allowing for partial goal failures.

Case-based planning (Hammond 1988; Alterman 1988) is primarily concerned with reusing plans from a library in order to increase the efficiency of plan generation. The key challenge is efficiently retrieving the most relevant plan and then adapting it to fit the current context. Our focus is on plan repair via merging, and the retrieval process (as well as the merging process) is based on expected utility.

Plan repair in classical planning (Tate 1977; Wilkins 1988) is concerned with finding replacements for "plan wedges" whose conditions have failed. Although our goal structure resembles plan wedges, we are concerned with finding a repair that maximizes the global utility, rather than concentrating on restoring conditions of the remainder of the plan. We expect that similar behavior would result in some cases, because the utility of a plan that can pick up execution with the remainder of the plan will often be higher than one that requires dropping large pieces of the plan suffix. However, the assumption, implicit in the classical planning approach, that the system can re-achieve the exact state originally described in the plan is, in general, not valid for a plan representation that describes temporal and dynamic state conditions.

Status and Open Issues

The first version of our rover executive was used on board the Marsokhod rover in an Ames field test in February, 1999 in the Mojave Desert. A new version of the executive was used on board the K9 rover in a joint JPL-Ames field test, held in May, 2000 in Nevada. Both of these versions supported flexible, contingent execution, but they did not update utilities nor use alternate plans. We have implemented prototype versions of the executive for alternate plans and utility-based execution. Our next steps will be to augment the plan language with goal structure constructs and to integrate the capabilities into the overall rover executive system.

This work raises a number of open issues. The interaction of branch points with action skipping can be complex. A branch point is a choice dictated by conditions on the state, resources, and time; thus, it may not be arbitrarily deleted from the plan. If a branch point does not depend on the skipped action, nor on prior actions deleted due to the skipped action, then the branch point is still valid. In this case, actions within any of the branches will be deleted if they are dependent on the skipped action. Action deletion can be performed independently on each branch, since there is no interaction across branches. On the other hand, when a branch point is dependent on a deleted action, then the conditions may no longer make sense; *e.g.*, consider a branch dependent on the outcome of a deleted experiment. For such cases, we are considering allowing branch-point deletion, using an annotation that indicates which branch should be followed (*i.e.*, replace the subtree). The correct treatment of this case will require careful consideration of the intended semantics of branch points in the presence of action deletion.

In addition, the ability to skip nodes and invoke alternate plans at multiple levels of the hierarchy introduces complexity into the execution semantics. One instance of this is that recovery plans need to be considered before the execution context is "popped" and lost; the comparison of expected utilities with and without recovery must take into account the full impact of changes on the execution context.

Another issue is that iteration is not explicitly handled in our framework. This design choice was made to simplify the task of constructing plans and evaluating their utility. Loops may be unrolled, but an accurate representation would require branches at the end of each iteration. This increases the amount of memory required to store the (redundant) plan steps for each iteration. The most elegant solution would be to represent explicitly the iteration, but this would require more complex planning and utility-evaluation techniques, perhaps moving towards Markov-model approaches.

We are also exploring other forms of on-board plan modification. We are collaborating on a project that is exploring Markov-model approaches to selecting among possible task decompositions, evaluating goal achievement, and potentially reordering goals to optimize plan quality (Bernstein *et al.* 2001; Zilberstein & Mouaddib 1999). This will allow more extensive plan modifications, while respecting the original plan goals.

Acknowledgments: We would like to acknowledge David E. Smith, Keith Golden, and Trey Smith for their contributions to the design of the Contingent Rover Language. We would also like to acknowledge the current K9 rover team for their support of rover operations and field tests: Kevin Bass, Maria Bualat, Larry Edwards, Lorenzo Flueckiger, Linda Kobayashi, and Anne Wright.

References

Alterman, R. 1988. Adaptive planning. *Cognitive Science* 12:393–421.

Bernstein, D.; Zilberstein, S.; Washington, R.; and Bresina, J. 2001. Planetary rover control as a markov decision process. In *Proceedings of the AAAI Spring Symposium: Game Theoretic and Decision Theoretic Agents.*

Bresina, J. L., and Washington, R. 2000. Expected utility distributions for flexible, contingent execution. In *Proceedings of the AAAI-2000 Workshop: Representation Issues for Real-World Planning Systems.*

Bresina, J.; Golden, K.; Smith, D. E.; and Washington, R. 1999. Increased flexibility and robustness for Mars rovers. In *In Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics, and Automation in Space.*

Grewal, M. S., and Andrews, A. P. 1993. *Kalman Filtering: Theory and Practice.* Prentice Hall.

Hammond, K. J. 1988. Case-based planning. In Kolodner, J., ed., *Proceedings of Workshop on Case-Based Reasoning*, 17–20.

Tate, A. 1977. Generating project networks. In *Proceedings of IJCAI-77*, 888–893. IJCAI.

Tsamardinos, I.; Pollack, M. E.; and Horty, J. F. 2000. Merging plans with temporal constraints, temporally extended actions, and conditional branches. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems.*

Washington, R.; Golden, K.; Bresina, J.; Smith, D. E.; Anderson, C.; and Smith, T. 1999. Autonomous rovers for mars exploration. In *Proceedings of The 1999 IEEE Aerospace Conference.*

Washington, R.; Golden, K.; and Bresina, J. 1999. Plan execution, monitoring, and adaptation for planetary rovers. In *In Proceedings of the IJCAI-99 Workshop: Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World.*

Wilkins, D. E. 1988. *Practical Planning.* Morgan Kaufman.

Yang, Q. 1997. *Intelligent Planning: A Decomposition and Abstraction Based Approach.* Springer.

Zilberstein, S., and Mouaddib, A.-I. 1999. Reactive control of dynamic progressive processing. In *Proceedings of IJCAI*, 1268–1273.